Everybody likes atomic generators. If you have a *vmm_data* class, all you have to do is add a quick macro line and you get a free VMM-compliant generator that you only need to instantiate in your environment. An atomic generator is a simple yet powerful means to create a continuous flow of random data items to your DUT (more accurately “ to your BFM). Once everything is hooked up, all you have to do in your test program is set the number of items you want to generate, or let it run endlessly.

For example, if you want to drive 100 items, do this:

{code}

program test;

my_env.gen.stop_after_n_insts = 100;

endprogram

{/code}

And if you want the generator to work endlessly do this:

{code}

```
program test;
 ...
 my_env.gen.stop_after_n_insts = 0;
 ...
 endprogram
```

{/code}

That's right, it's a zero up there, and it's also the default. So if you don't say anything, all of your atomic generators will work endlessly. So, one might ask, how do you turn it off? Well, seems like the only way to do this is to call the *stop_xactor()* method, but only if you call it immediately after *vmm_env::start()*. This means that if your test program looks like this, both generators will still work although *stop_xactor()* is called:

{code}

```
program test;
```

```
env = new();
```

```
env.build();

env.gen1.stop_after_n_insts = 100;

env.gen1.stop_xactor();

env.gen2.stop_after_n_insts = 0;

env.gen2.stop_xactor();

env.run();

endprogram
```

{/code}

The generators will stop, however, if you place the *stop_xactor()* after explicitly calling the *start( )*

method:

{code}

program test;

env = new();

env.build();

env.gen1.stop_after_n_insts = 100;

env.gen2.stop_after_n_insts = 0;

env.start();

env.gen1.stop_xactor();

env.gen2.stop_xactor();

env.run();

endprogram

{/code}

Now, although this is the recommended way by Synopsys we just couldn't resist hacking the code to make *stop_after_n_insts = 0* do NOTHING. Be careful though, this hack implies a minor paradigm shift - atomic generators are now off by default (rather than on and working endlessly). With the hacked version, you can do this in your test, no need to *stop_xactor()*
:

{code}

program test;

// choose only one of the following:

my_env.gen1.stop_after_n_insts = 0; // shut down (default)

my_env.gen1.stop_after_n_insts = 100; // create 100 items and then stop

my_env.gen1.top_after_n_insts = -1; // work endlessly

endprogram

{/code}

THE HACK

WARNING:

THIS IS FOR ADVANCED VMM USERS ONLY. USE AT YOUR OWN RISK!

KIDS, DON'T TRY THIS AT HOME.

1. **Locate the following line in the VMM source code:**

   {code}class `vmm_atomic_gen_(class_name) extends `VMM_XACTOR; {/code}

2. **About 100 lines down from there replace this line:**

   {code}while (this.stop_after_n_insts <= 0 ||{/code}

   with this one:

   {code}while (this.stop_after_n_insts < 0 ||{/code}

3. **There's a similar code line in the vmm_scenario class definition, make sure you haven't ruined that one by mistake.**
4. **Remember: This is only a small change, but it has a big impact. Make sure you understand the paradigm shift.**